```
%% Specify Crystal and Specimen Symmetries
clear all
close all

CS = {'notIndexed',...
  crystalSymmetry('622', [3.2 3.2 5.2], 'X||a', 'Y||b*', 'Z||c', 'mineral', 'Mg',↙
'color', 'light blue')};

% plotting conventions
setMTEXpref('xAxisDirection','east');
setMTEXpref('zAxisDirection','intoPlane');

%% Specify File Names
% which files to be imported
% fname = [pname '\0.08strain.ang'];0.08strain_minimalCleanup.ang
fname = ['map20170109123448657.ang'];

%% Import the Data

% create an EBSD variable containing the data
ebsd = loadEBSD(fname,CS,'interface','ang',...
  'convertEuler2SpatialReferenceFrame');
ebsd(ebsd.prop.iq==0).phaseId=1;
ebsd=ebsd(ebsd.prop.ci>0.1);

oM = ipdfHSVOrientationMapping(ebsd('indexed'));
color = oM.orientation2color(ebsd('indexed').orientations);

seg_angle = 15;

% minimum indexed points per grain between 5 and 10
min_points = 5;

[grains,ebsd.grainId,ebsd.mis2mean] = calcGrains(ebsd('indexed'),...
    'angle',seg_angle*degree,'boundary','tight');

% remove small grains with less than min_points indexed points
figure;plot(grains('Mg'),grains('Mg').meanOrientation)

grainsSelected = grains(grains.grainSize > 5);

%reindex ebsd
ebsd = ebsd(grainsSelected);
% figure;plot(grains('Mg'),grains('Mg').meanOrientation)

% and perform grain reconstruction with the reduces EBSD data set
[grains,ebsd.grainId,ebsd.mis2mean] = calcGrains(ebsd('indexed'),'angle',5*degree);

figure;plot(grains('Mg'),grains('Mg').meanOrientation)
%% Get pairs for twin relationships using mean orientation
```

```
tol=8*degree; %needs to be slight larger (like Rod noted) because dealing with meanOrientations

% We are going to use pairs to store all pairs information.
[counts,pairsTmp] = neighbors(grains);
pairs=struct;
pairs.type=zeros(length(pairs),1,'int8');
pairs.combine=zeros(length(pairs),1,'logical');
pairs.pairs=pairsTmp
pairs.grainId1=pairsTmp(:,1);
pairs.grainId2=pairsTmp(:,2);
pairs.len=length(pairsTmp);

%Specify the twin definitions
tt1 = orientation('axis',Miller(1,-2,1,0,ebsd{2}.CS,'uvw'),...
  'angle',86.3471*degree,ebsd{2}.CS,ebsd{2}.CS); %Extension twin <10-11> largest amount
tt2 = orientation('axis',Miller(1,0,-1,0,ebsd{2}.CS,'uvw'),...
  'angle',34.7*degree,ebsd{2}.CS,ebsd{2}.CS); %Extension twin <1126> Quite a few
ct1 = orientation('axis',Miller(1,0,-1,0,ebsd{2}.CS,'uvw'),...
  'angle',64.3*degree,ebsd{2}.CS,ebsd{2}.CS); %Main contraction twin

%check if pairs contain twins
for i=1:pairs.len
    mori=inv(grains(pairs.grainId1(i)).meanOrientation)*grains(pairs.grainId2(i)).meanOrientation; %Compute misorientation between all pairs
    [pairs.combine(i),pairs.type(i)] = TestTwinRelationship(ebsd,mori,tol,tt1,tt2,ct1);
end
pairs.grainId1_TC=pairs.grainId1(pairs.combine)
pairs.grainId2_TC=pairs.grainId2(pairs.combine)
pairs.type_TC=pairs.type(pairs.combine)
pairs.pairs_TC=pairs.pairs(pairs.combine,1:2)
grains.prop.boundaryType=pairs.type

% grains.prop.type(pairs(combine,1:2))
% figure;plot(grains(pairs(combine,1:2)))
%% Compute schmid factor for the various twin combinations (Complete) Moving to in grain calculations since I care about pairs and mapping is a pain.
CRSS_TT1=111;
CRSS_TT2=113;
CRSS_CT1=195;
sigma = tensor([0 0 0; 0 0 0; 0 0 -1],'name','stress') %Sign of loading does more than just invert twin/parent flag

subgrains1 = grains(pairs.grainId1); %could just pass pairs
subgrains2 = grains(pairs.grainId2);
subgrains1.prop.type=pairs.type;
subgrains2.prop.type=pairs.type;

[sFRelative12,sF1,sF2,SFActiveVar] = GetSchmidRelative(subgrains1,subgrains2,tt1,tt2,ct1,CRSS_TT1,CRSS_TT2,CRSS_CT1,sigma);
```

```
%sF is stored for pairs where 1 and 2 are for each. Right now not computing
%none twin sF
pairs.sF1=sF1';
pairs.sF2=sF2';
pairs.sFRelative12=sFRelative12';
pairs.sFActiveVar=SFActiveVar;
pairs.sF1_TC=pairs.sF1(pairs.combine);
pairs.sF2_TC=pairs.sF2(pairs.combine);
grains.prop.sF1=pairs.sF1;
grains.prop.sF2=pairs.sF2;
grains.prop.sFRelative12=pairs.sFRelative12;
grains.prop.sFActiveVar=pairs.sFActiveVar;
%plot schmid relationships. relationships can plot over each other because not grouped
%yet.
% plotType=2;
% figure
% plot(subgrains1(subgrains1.prop.type==plotType),pairs.sF1_TC(pairs.type_TC==plotType))
% hold on
% plot(subgrains2(subgrains1.prop.type==plotType),pairs.sF2_TC(pairs.type_TC==plotType))
% hold off

%% Construct intergranular graph
s=pairs.pairs(:,1)';
t=pairs.pairs(:,2)';
G=graph(s,t);

%Comments: Once we have all the pertinent data in the graph we no longer
%have to worry about indexing and mapping between grains and merged grains
%etc... This is the primary reason that I chose the graph method!

% Add node labels and other grain level information
G.Nodes.Name=cellstr(int2str(grains.id))
G.Nodes.Id=str2num(cell2mat(G.Nodes.Name));
G.Nodes.Area=grains.area;
G.Nodes.Perimeter=grains.perimeter;
G.Nodes.AspectRatio=grains.aspectRatio;
G.Nodes.Paris=grains.calcParis;
G.Nodes.centroids=grains.centroid;
G.Nodes.meanOrientation=grains.meanOrientation
G.Nodes.Properties.UserData.mineral=grains.mineral; %For single phase material

% Compute grain boundary for each grain fragment (i.e. node)
G.Nodes.Gb=cell(length(grains),1);

for i=1:length(grains)
    G.Nodes.Gb{i}=grains(i).boundary;
end

% Add intergranular information
G.Edges.pairs=pairs.pairs;
G.Edges.type=pairs.type'; %Twin relation type
```

```
G.Edges.SF=[pairs.sF1,pairs.sF2]; %Schmid factors for relation type
G.Edges.SFRelative12=pairs.sFRelative12;
%Want to filter out SFRelative values that come from schmid factors with same sign?
%use the voting scheme
% G.Edges.SFRelative12(sum(sign(G.Edges.SF(:,1:2)),2)==2)=0
G.Edges.SFActiveVar=pairs.sFActiveVar';
% G.Edges.disorientation=

% Make a list of boundaries shared between two grains connected by an edge
G.Edges.Gb=cell(pairs.len,1);
G.Edges.ebsdId=cell(pairs.len,1);

grain1=grains(pairs.grainId1);
grain2=grains(pairs.grainId2);
for i=1:length(grain1)
    id1=grain1(i).boundary.ebsdId;
    id2=grain2(i).boundary.ebsdId;
    [boundaryEbsdId,loc]=intersect(id1,id2,'rows');
    G.Edges.Gb{i}=grain1.boundary(loc);
    G.Edges.ebsdId{i}=boundaryEbsdId;
end

%Overlayer graph on grains
figure;
h=plot(grains,grains.meanOrientation,'Micronbar','off')
set(gca,'Units','normalized');
savefig('grains.fig');
hold on
p=plot(G,'XData',G.Nodes.centroids(:,1),'YData',G.Nodes.centroids(:,2))
hold off
p.Marker='s';p.NodeColor='k';p.MarkerSize=3;p.EdgeColor='k';



%% Remove nodes that have grains that are too small
% G=rmnode(G,grains(grains.grainSize < 5).id)
%% Reduce intergranular graph to grain clusters
%(i.e. delete edges between neighboring grain clusters)

%Remove edges that don't satisfy the twin relationship
G.Edges.Removed=G.Edges.type==0;
G_clust=rmedge(G,G.Edges.pairs(G.Edges.Removed,1),...
    G.Edges.pairs(G.Edges.Removed,2))

%Using the gplot gui editor to delete unwanted connections
% A=adjacency(G_clust);
% xy=G_clust.Nodes.centroids;
% graph_gui(A,xy,grains)

% Overlayer graph on grains
figure;
```

```
plot(grains,grains.meanOrientation)
hold on
p=plot(G_clust,'XData',G_clust.Nodes.centroids(:,1),'YData',G_clust.Nodes.centroids(:,2))
hold off
p.Marker='s';p.NodeColor='k';p.MarkerSize=3;p.EdgeColor='k';
labeledge(p,1:length(G_clust.Edges.pairs),1:length(G_clust.Edges.pairs))

%% Remove edges from list generated visually (improve on this later
edgeList2Remove=[]; % number of with respect to the graph above
G_clustClean=rmedge(G_clust,edgeList2Remove)

figure;
plot(grains,grains.meanOrientation)
hold on
p=plot(G_clustClean,'XData',G_clustClean.Nodes.centroids(:,1),'YData',G_clustClean.Nodes.↙
centroids(:,2))
hold off
p.Marker='s';p.NodeColor='k';p.MarkerSize=3;p.EdgeColor='k';
labeledge(p,1:length(G_clustClean.Edges.pairs),1:length(G_clustClean.Edges.pairs))

%% Identify Families in grain clusters
%Remove nodes that aren't connected by edges, alternatively use condition
%on minimum occurance from conncomp
Grains2Keep=unique(G_clustClean.Edges.pairs)
G_clustClean=rmnode(G_clustClean,G_clustClean.Nodes.Id(~ismember(G_clustClean.Nodes.Id,↙
Grains2Keep)));

G_clustClean.Nodes.Group = conncomp(G_clustClean)'; %Get the nodes that are connected by↙
edges


%Asign each pair to a cluster of grains
G_clustClean.Edges.Group=zeros(length(G_clustClean.Edges.pairs),1);
for i=1:length(G_clustClean.Edges.pairs)
      G_clustClean.Edges.Group(i)=G_clustClean.Nodes.Group(...
          find(G_clustClean.Edges.pairs(i,1)==G_clustClean.Nodes.Id));
end

%loop over grain clusters to determine parent, grain groups
G_clustClean.Nodes.FamilyID=zeros(length(G_clustClean.Nodes.Id),1);
for i=1:max(G_clustClean.Edges.Group)
    egroupId= find((i==G_clustClean.Edges.Group)==true); %convert logical arrays to↙
indices
    ngroupId= find((i==G_clustClean.Nodes.Group)==true);

    ori=G_clustClean.Nodes.meanOrientation(ngroupId);
    G_clustClean.Nodes.FamilyID(ngroupId)=GetFamily(ori)
end

%Determine what family each pair relates
G_clustClean.Edges.FamilyID=zeros(length(G_clustClean.Edges.pairs),2);
```

```
for i=1:max(G_clustClean.Edges.Group)
    egroupId= find((i==G_clustClean.Edges.Group)==true); %convert logical arrays to↙
indices
    ngroupId= find((i==G_clustClean.Nodes.Group)==true);
    nId=G_clustClean.Nodes.Id(ngroupId);
    fId=G_clustClean.Nodes.FamilyID(ngroupId);
    for j=1:length(egroupId)
        G_clustClean.Edges.FamilyID(egroupId(j),1)=unique(fId(G_clustClean.Edges.pairs↙
(egroupId(j),1)==nId));
        G_clustClean.Edges.FamilyID(egroupId(j),2)=unique(fId(G_clustClean.Edges.pairs↙
(egroupId(j),2)==nId));
    end
end

figure;
plot(grains(G_clustClean.Nodes.Id),G_clustClean.Nodes.FamilyID,'Micronbar','off')
hold on
p=plot(G_clustClean,'XData',G_clustClean.Nodes.centroids(:,1),'YData',G_clustClean.Nodes.↙
centroids(:,2))
hold off
p.Marker='s';p.NodeColor='k';p.MarkerSize=3;p.EdgeColor='k';
labeledge(p,1:length(G_clustClean.Edges.pairs),1:length(G_clustClean.Edges.pairs))

%% Implement voting scheme for the respective families
% For each family compute the vote between connected families
%Edges currently connect clustered grains. We can compute the vote for each
%edge and then average for the family. This ensures that only neighbors are
%ever voted on. So that votes are really between familys average values of
%schmid are computed and stored for each fragment in a family. The area and
%relative boundary are taken in the addative sense rather than average.

%Edges.Gb contains boundaries between pairs
%Nodes.Gb contains boundaries for each fragment
%To compute the relative boundary between families we need to sum boundary
%of a particular type and since each pair is its own type this should be
%relatively simple.
w=[1,1,1]
G_clustClean.Nodes.Properties.UserData.Mineral=grains.mineral;
G_Complete = FamilyVotes(G_clustClean,w);


%%
G_Complete.Nodes.isTwin=zeros(length(G_Complete.Nodes.Id),1);
G_Complete.Nodes.isAParent=zeros(length(G_Complete.Nodes.Id),1,'logical');
for i=1:max(G_Complete.Edges.Group)
    egroupId= find((i==G_Complete.Edges.Group)==true); %converts logical arrays to↙
indices
    ngroupId= find((i==G_Complete.Nodes.Group)==true);
    nFamily = G_Complete.Nodes.FamilyID(ngroupId)
    nId = G_Complete.Nodes.Id(ngroupId)
    eType = G_Complete.Edges.type(egroupId)
```

```
    eVote = G_Complete.Edges.Vote(egroupId,:)
    ePairs = G_Complete.Edges.pairs(egroupId,:)
    eFamily = G_Complete.Edges.FamilyID(egroupId,:)

    %Initialize parent
    Parent = zeros(size(ePairs,1),2,'logical');

    %Make list of each family relation
    FamilyRelationList=cell(max(nFamily),1);
    for j=1:max(nFamily)
        FamilyInPair=ismember(eFamily(:,:),j)
        FamilyRelationList{j}=FamilyInPair
        FamilyRelationList{j}
        eFamily
    end

    %The number of relation types for each family
    %Here we also assign the parent for the case of two tensile variants
    %having the same family.
    numTypeFamily=zeros(max(nFamily),1);
    for j=1:max(nFamily)
        id=logical(sum(ismember(eFamily(:,:),j),2))
        uniqueTypes=unique(eType(id))
        numTypeFamily(j)=length(uniqueTypes)
        if numTypeFamily(j)>1
            if ismember([1,2],uniqueTypes) %tensile twins can't make tensile twins
                Parent(FamilyRelationList{j})=true;
            end
        end
    end

    %Assign the rest of the parents
    for j=1:size(Parent,1)
        if sum(Parent(j,:))==0
            [~,loc]=max(eVote(j,:));
            Parent(j,loc)=true;
        end
    end
    %Finally assign twin label to nodes (need to differentiate generation)
    G_Complete.Nodes.isTwin(ngroupId(ismember(nId,ePairs(~Parent))))=5; eType
    G_Complete.Nodes.isAParent(ngroupId(ismember(nId,ePairs(Parent))))=true ;
end

figure;
plot(grains(G_Complete.Nodes.Id),G_Complete.Nodes.isTwin,'Micronbar','off')
figure;
plot(grains(G_Complete.Nodes.Id),grains(G_Complete.Nodes.Id).↙
meanOrientation,'Micronbar','off')

%%
```

```
        ori1=G_clustClean.Nodes.meanOrientation(pairId(j,1));
        ori2=G_clustClean.Nodes.meanOrientation(pairId(j,1));
figure;
plot(grains(333),grains(333).meanOrientation,'Micronbar','off')
hold on
p=plot(G_clustClean,'XData',G_clustClean.Nodes.centroids(:,1),'YData',G_clustClean.Nodes.↙
centroids(:,2))
hold off
p.Marker='s';p.NodeColor='k';p.MarkerSize=3;p.EdgeColor='k';
labeledge(p,1:length(G_clustClean.Edges.pairs),1:length(G_clustClean.Edges.pairs))


%% Now we want to merge the grains together into sets
%Build a list of boundaries to merge
gBList=G_clustClean.Edges.Gb{1}('Mg','Mg');
for i=2:length(G_clustClean.Edges.pairs)
    gBList=[BoundaryList;G_clustClean.Edges.Gb{i}('Mg','Mg')];
end
[mergedGrains,parentId] = merge(grains,gBList('Mg','Mg'));

% copy ebsd data into a new variable to not change the old data
ebsd_merged = ebsd;
% update the grainIds to the parentIds
ebsd_merged('indexed').grainId = parentId(ebsd('indexed').grainId)
figure
plot(grains,grains.meanOrientation)
hold on
plot(mergedGrains.boundary,'k','linewidth',4)
plot(grains.boundary,'linecolor','r','linewidth',1)
text(grains,int2str(grains.id))
hold off

[MergedGrains,ParentId]=merge(grains,BoundaryList);

% copy ebsd data into a new variable to not change the old data
ebsd_merged = ebsd;
% update the grainIds to the parentIds
ebsd_merged('indexed').grainId = parentId(ebsd('indexed').grainId)
figure
plot(grains,grains.meanOrientation)
hold on
plot(mergedGrains.boundary,'k','linewidth',4)
plot(grains.boundary,'linecolor','r','linewidth',1)
text(grains,int2str(grains.id))
hold off

%% Get boundaries between EBSD points
[twinBoundary_tt1,twinBoundary_tt2,twinBoundary_ct1,twinBoundary_sec1,↙
twinBoundary_ttwint] = GetTwinBoundaries(ebsd,grains.boundary('Mg','Mg'),5*degree);
combinedTwinTot=[twinBoundary_tt1;twinBoundary_tt2;twinBoundary_ct1] %twinBoundary_tt1;↙
```

```
twinBoundary_tt2;twinBoundary_ct1;

%% Let's see how we have done
figure
plot(ebsd,ebsd.orientations)
hold on
% plot(gB('Mg','Mg'),'linecolor','k','linewidth',6)
% plot(combinedTwinTot,'linecolor','r','linewidth',2)
plot(twinBoundary_tt1,'linecolor','r','linewidth',2)
plot(twinBoundary_tt2,'linecolor','b','linewidth',2)
plot(twinBoundary_ct1,'linecolor','g','linewidth',2)
% plot(twinBoundary_sec1,'linecolor','b','linewidth',2)
text(grains,int2str(grains.id))
% scatter(boundaryList(:,1),boundaryList(:,2))
hold off

%% Add and remove grain pairs that are not likely to be twins using previous graph
pairs.toRemove=[103,241;299,349;398,440;511,440;441,511;606,505;...
    470,429;290,225;312,328;682,683;658,641;410,366;636,592;649,600;...
    646,593;168,203;270,211;306,360;287,360;275,360;306,328;360,293;211,207;...
    304,306;308,306];
pairs.toAdd=[];%168,203 Need to search for the pair in pairs.pairs. It won't have a↙
schmid factor so set type and schmid factor to 0;

pairs.combineClean=pairs.combine;
pairs.typeClean=pairs.type;

%make all pair combinations and make sure they can't/should be combined
pairs.toRemove=[pairs.toRemove;fliplr(pairs.toRemove)];
[boundaryLoc,loc]=intersect(pairs.pairs,pairs.toRemove,'rows');
pairs.combineClean(loc)=false;
pairs.typeClean(loc)=0;
pairs.pairsClean=[pairs.pairs];%;pairs.toAdd];
pairs.combineClean=[pairs.combineClean];%;ones(size(pairs.toAdd,1),1,'logical')];

%Recalculate pairs for the modified data
pairs.pairs_TCClean=pairs.pairsClean(pairs.combineClean,1:2);
pairs.grainId1_TCClean=pairs.pairsClean(pairs.combineClean,1);
pairs.grainId2_TCClean=pairs.pairsClean(pairs.combineClean,2);
pairs.type_TCClean=pairs.typeClean(pairs.combineClean);
pairs.sF1_TC=pairs.sF1(pairs.combineClean);
pairs.sF2_TC=pairs.sF2(pairs.combineClean);

% Redo calculation from new pairs list
boundaryList=[];
grain1=grains(pairs.pairsClean(pairs.combineClean,1));
grain2=grains(pairs.pairsClean(pairs.combineClean,2));
for i=1:length(grain1)
    id1=grain1(i).boundary.ebsdId;
    id2=grain2(i).boundary.ebsdId;
    [boundaryLoc,loc]=intersect(id1,id2,'rows');
```

```
%     gBtmp=grain1.boundary(loc);

    boundaryList=[boundaryList;boundaryLoc];
end

id=grains.boundary.ebsdId;
[boundaryLoc,loc]=intersect(id,boundaryList,'rows');
gB_clean=grains.boundary(loc)
figure
plot(grains,grains.meanOrientation)
hold on
plot(gB_clean('Mg','Mg'),'linecolor','k','linewidth',6)
% plot(combinedTwinTot,'linecolor','r','linewidth',2)
plot(twinBoundary_tt1,'linecolor','r','linewidth',2)
plot(twinBoundary_tt2,'linecolor','b','linewidth',2)
plot(twinBoundary_ct1,'linecolor','g','linewidth',2)
% plot(twinBoundary_sec1,'linecolor','b','linewidth',2)
text(grains,int2str(grains.id))
% scatter(boundaryList(:,1),boundaryList(:,2))
hold off

%% Merge grains into clusters for twin/parent analysis
[mergedGrains,parentId] = merge(grains,gB_clean('Mg','Mg'));

% copy ebsd data into a new variable to not change the old data
ebsd_merged = ebsd;
% update the grainIds to the parentIds
ebsd_merged('indexed').grainId = parentId(ebsd('indexed').grainId)
figure
plot(grains,grains.meanOrientation)
hold on
plot(mergedGrains.boundary,'k','linewidth',4)
plot(grains.boundary,'linecolor','r','linewidth',1)
text(grains,int2str(grains.id))
hold off
%Good to here. Now we need to segment the combined grains.

% From here on we should be dealing with the merged grain set?

%% Create Families
%plot the mergedGrains IDs
figure
plot(grains,grains.meanOrientation)
hold on
plot(mergedGrains.boundary,'k','linewidth',4)
plot(grains.boundary,'linecolor','r','linewidth',1)
text(mergedGrains,int2str(mergedGrains.id))
hold off
grains.prop.familyID=zeros(length(grains),1,'int8');
%%
%For schmid calculations
```

```
CRSS_TT1=111; %from cpfe
CRSS_TT2=113;
CRSS_CT1=195;
sigma = tensor([0 0 0; 0 0 0; 0 0 -1],'name','stress') %Sign of loading does more than
just invert twin/parent flag

grains.prop.familyID=zeros(length(grains),1,'int8');
grains.prop.sF1=zeros(length(grains),1,'int8');
grains.prop.sF2=zeros(length(grains),1,'int8');



grains.prop.grainId1=zeros(length(grains),1,'int8');
grains.prop.grainId2=zeros(length(grains),1,'int8');
% grains.prop.grainId1=grains(ismember(grains.id,pairs.grainId2)

groups=struct;
groups.len=length(mergedGrains);

groups.family=cell(groups.len,1)
groups.grains={}
% groups.combine=zeros(length(groups),1,'logical');
% groups.groups=groupsTmp
% groups.grainId1=groupsTmp(:,1);
% groups.grainId2=groupsTmp(:,2);
% groups.len=length(groupsTmp);

for i=184%1:length(mergedGrains)
    groups.grains{i}=grains(parentId==mergedGrains(i).id);
    [counts2,pairs2]=neighbors(groups.grains{i})
    figure;plot(groups.grains{i},groups.grains{i}.meanOrientation)
    %group similar orientations
    len=length(grainCluster);
    oriSame=zeros(len,len,'logical');
    FamiliesComplete=zeros(len,1,'logical');
    cnt=0;
    for j=1:len
        for k=1:len
            oriSame(j,k)=angle(grainCluster(j).meanOrientation,grainCluster(k).
meanOrientation) /degree <10;
        end

        if FamiliesComplete(oriSame(j,:))~=true
            cnt=cnt+1;
            FamiliesComplete(oriSame(j,:))=true;
            grains.prop.familyID(grainCluster.id(oriSame(j,:)))=cnt;
        end
        if sum(FamiliesComplete)==len
            break;
        elseif sum(FamiliesComplete)==len-1
            cnt=cnt+1;
```

```
            grains.prop.familyID(grainCluster.id(~FamiliesComplete))=cnt;
            break;
        end
    end

    %Compute grain family properties
    for j=1:cnt
        family=grains.prop.familyID(grainCluster.id)==j;
        grains.prop.familyGbLen(grainCluster.id(family))=grainCluster(family).↙
boundarySize; %boundary lengths of families
        grains.prop.familySize(grainCluster.id(family))=grainCluster(family).area;
        %Add schmid here?
%         [countsFamilies,pairsFamilies]=neighbors(grains(grainCluster.id));
%         m(j)=mean(ebsd.orientations(ismember(ebsd.grainId,grains.id(grainCluster.id↙
(family)))))
    end
        %get pairs inside merged grain
        id=(ismember(pairsClean(combineClean,1),grainCluster.id)+...
            ismember(pairsClean(combineClean,2),grainCluster.id)); %PairsClean consists↙
of grain ids
        pairID=find(id==2) %Should be the indices of pairs that are in our combined grain

        %Determine family pair and direction for each schmid pair
        pairFamily=cell(cnt,1);
        for j=1:cnt
            family=grains.prop.familyID(grainCluster.id)==j;
            pairFamily{j}=[ismember(pairsClean(combineClean(pairID),1),grainCluster.id↙
(family)),...
                ismember(pairsClean(combineClean(pairID),2),grainCluster.id(family))]; %↙
this indexes pairsClean
        end

        familyFs=grains.prop.sF1(ismember(grains.id,pairsClean(pairFamily{1},1)))
        grains.prop.sF2(ismember(grains.id,pairsClean(pairFamily{1},1)))
        grains.prop.sF2(ismember(grains.id,pairsClean(pairFamily{3},1)))

        grains.prop.sF2(pairID)
        intersect(pairFamily{1},pairFamily{2})

        subgrains1 = grains(pairs(combine,1));
        subgrains2 = grains(pairs(combine,2));
        subgrains1.prop.type=grains.prop.type(combine);
        subgrains2.prop.type=grains.prop.type(combine);

        [sF1,sF2] = GetSchmidRelative(subgrains1,subgrains2,tt1,tt2,ct1,CRSS_TT1,↙
CRSS_TT2,CRSS_CT1,sigma);
        grains.prop.sF1=zeros(length(grains),1,'int8');
        grains.prop.sF2=zeros(length(grains),1,'int8');
        grains.prop.sF1(combine)=sF1';
        grains.prop.sF2(combine)=sF2';
```

```
        %plot schmid relationships. relationships can overright because not grouped
        %yet.
        plotType=subgrains1.prop.type==1;
        figure
        plot(subgrains1(plotType),sF1(plotType))
        hold on
        plot(subgrains2(plotType),sF2(plotType))
        hold off




%         grains.prop.familySFtt1(grainCluster.id(family))=grains.prop.sF1(grainCluster.↙
id(family))
%         grains.prop.familySFtt2
%         grains.prop.familySFct1
end

% end
% grains.prop.familyID(grainCluster.id)
figure
plot(grains,grains.prop.familyID)
hold on
plot(mergedGrains.boundary,'k','linewidth',4)
hold off

%% Compute boundary lengths of families


%% Now we need to make a list of boundaries shared between the grains
boundaryList=[];
grain1=grains(pairs(combine,1));
grain2=grains(pairs(combine,2));
for i=1%:length(grain1)
    x1=grain1(i).boundary.x;
    y1=grain1(i).boundary.y;
    x2=grain2(i).boundary.x;
    y2=grain2(i).boundary.y;
    [boundaryLoc,loc]=intersect([x1,y1],[x2,y2],'rows');
    gB=grain1.boundary(loc);
    if i==1
%         boundaryList=gB('Mg','Mg');
        boundaryList=boundaryLoc;
    else
%         boundaryList=[boundaryList;gB('Mg','Mg')];
        boundaryList=[boundaryList;boundaryLoc];
    end
    % scatter(boundarLoc(:,1),boundarLoc(:,2))
%     plot(grain1.boundary(loc),'k','linewidth',3)
%     hold off
```

```
%     [mergedGrains,parentId] = merge(grains,gB('Mg','Mg'));
end
%Since this clearly works
x=grains.boundary.x; y=grains.boundary.y;
[boundaryLoc,loc]=intersect([x,y],boundaryList,'rows');
xy=[grains.boundary.x(loc),grains.boundary.y(loc)];
%works for xy but xy location index ~=boundary index WHY!!!!
%segmentIds are meaningless..
figure
plot(ebsd('Mg'),ebsd('Mg').orientations)
hold on
plot(gB,'k','linewidth',3)
% scatter(boundaryList(:,1),boundaryList(:,2))
scatter(xy(:,1),xy(:,2),'k')
hold off




%%

plot(grains)
hold on
plot(grains(pairs(163,1:2)),'yellow')
hold off


[mergedGrains,parentId] = merge(grains,combinedTwin);
% copy ebsd data into a new variable to not change the old data
ebsd_merged = ebsd;
% update the grainIds to the parentIds
ebsd_merged('indexed').grainId = parentId(ebsd('indexed').grainId)
figure
plot(ebsd_merged('Mg'),ebsd('Mg').orientations)
hold on
plot(mergedGrains.boundary,'k','linewidth',3)
hold off



% hold on
% plot(grains2merge('Mg','Mg'),'linewidth',3)
% hold off


% for i=1:numGrains
%     if ~ismember(i,processNeighbors)
%         cnt=cnt+1;
%         processNeighbors(cnt)=i;
%     end
```

```
%
%     while length(processNeighbors)<=cnt
%         [counts,pairs] = neighbors(grains(grainId))
%         group=zeros(counts,1,'logical');
%         for j=1:counts
%             mis=angle(grains(i).meanOrientation,grains(pairs(j,2)).meanOrientation) /↙
degree
%             twinDected = TestTwinRelationship(ebsd,mis,tol)
%             if twinDetected
%
%             end
%         end
% end

gB = grains.boundary
gB_MgMg = gB('Mg','Mg')


% figure
plot(grains('Mg'),grains('Mg').meanOrientation)
% plot(ebsd('Mg'),ebsd('Mg').orientations)
hold on
% plot the boundary of all grains
plot(grains.boundary,'linewidth',4)
hold off

%% Get twin boundaries
[twinBoundary_tt1,twinBoundary_tt2,twinBoundary_ct1,twinBoundary_sec1,↙
twinBoundary_ttwint] = GetTwinBoundaries(ebsd,gB_MgMg,5*degree);

% plot the twinning boundaries
% figure
plot(grains('Mg'),grains('Mg').id(randperm(length(grains('Mg')))))
% plot(ebsd,ebsd.orientations)
hold on
%plot(gB_MgMg,angle(gB_MgMg.misorientation,twinning),'linewidth',4)
plot(twinBoundary_tt1,'linecolor','w','linewidth',2,'displayName','Tensile Twin 1↙
Boundary')
plot(twinBoundary_tt2,'linecolor','w','linewidth',2,'displayName','Tensile Twin 2↙
Boundary')
plot(twinBoundary_ct1,'linecolor','w','linewidth',2,'displayName','Compression Twin 1↙
Boundary')
plot(twinBoundary_sec1,'linecolor','w','linewidth',2,'displayName','Secondary Twin 1↙
Boundary')
plot(twinBoundary_ttwint,'linecolor','w','linewidth',2,'displayName','Tensile-Tensile↙
Twin Boundary')
text(grains,int2str(grains.id))
hold off

legend off
```

```
%% Merge grains and label the parent grains
combinedTwinTot=[twinBoundary_tt1;twinBoundary_tt2;twinBoundary_sec1;twinBoundary_ttwint]↙
%;twinBoundary_ct1 excluded because easy to identify
combinedTwin=combinedTwinTot;

%% Compute the misorientation between mean orientations to segment the microstructure



%% Need to remove twin boundary for merging that lies on parent grain boundaries.
numGrains=grains.length
ratio_gBs=zeros(numGrains,1)
for i=1:numGrains

    %Check if is x,y against all boundaries
    %If a twin has a twin boundary relationship with


    grainIdGlobal=or(gB_MgMg.grainId(:,1)==i,gB_MgMg.grainId(:,2)==i);
    grainIdTwin=or(combinedTwin.grainId(:,1)==i,combinedTwin.grainId(:,2)==i);
    tmp_gB=gB_MgMg(grainIdGlobal);
    tmp_gB_isTwinning=combinedTwin(grainIdTwin);

    ratio_gBs(i)=length(tmp_gB_isTwinning)/length(tmp_gB);
    if ratio_gBs(i)<0.0 && ratio_gBs(i) > 0
        combinedTwin(grainIdTwin)=[];
    end
end


% figure
% plot(grains)
% hold on
% % plot the boundary of all grains
% plot(,'linewidth',4)
% % plot(combinedTwin(grainIdTwin),'linewidth',4,'linecolor','w')
% hold off



[mergedGrains,parentId] = merge(grains,combinedTwin);
% copy ebsd data into a new variable to not change the old data
ebsd_merged = ebsd;
% update the grainIds to the parentIds
ebsd_merged('indexed').grainId = parentId(ebsd('indexed').grainId)
figure
plot(ebsd_merged('Mg'),ebsd('Mg').orientations)
hold on
plot(mergedGrains.boundary,'k','linewidth',3)
hold off
[twinBoundary_tt1,twinBoundary_tt2,twinBoundary_ct1,twinBoundary_sec1,↙
```

```
twinBoundary_ttwint] = GetTwinBoundaries(ebsd,gB_MgMg,5*degree);
[mergedGrains,parentId] = merge(mergedGrains,combinedTwinTot);
% plot the merged grains
%plot(ebsd,ebsd.orientations)
figure
plot(mergedGrains.boundary,'linecolor','k','linewidth',2.5,'linestyle','-',...
  'displayName','merged grains')
hold on
text(mergedGrains,int2str(mergedGrains.id))
plot(twinBoundary_tt1,'linecolor','r','linewidth',2,'displayName','Tensile Twin 1')
plot(twinBoundary_tt2,'linecolor','g','linewidth',2,'displayName','Tensile Twin 2')
plot(twinBoundary_ct1,'linecolor','b','linewidth',2,'displayName','Compression Twin')
plot(twinBoundary_sec1,'linecolor','y','linewidth',2,'displayName','Secondary Twin')
plot(twinBoundary_ttwint,'linecolor','c','linewidth',2,'displayName','Tensile-Tensile↙
Twin Boundary')
hold off
legend off

%% Loop over grains and pull twins out by their size in pixels
% gBInner=mergedGrains.innerBoundary;
% gBPar
% isTwinning = angle(gBInner.misorientation,tt1) < 7.5*degree;
% twinBoundaryInner=gBInner(isTwinning)

%Note: the parent ID maps each grain in original scan to a merged grain.
%We want to to extract all of the grains within a merged grain. Finally we
%Want to reform the parent grain and visualize the result to make sure the
%algorithm is doing what we want.

% Make grain id map and compare to IPF map
% figure
% plot(grains,grains.id(randperm(length(grains))),'micronbar','off','figSize','large')
% hold on
% plot(grains.boundary)
% mtexColorMap hsv
% hold off

% color = oM.orientation2color(ebsd.orientations);
% figure;
% plot(ebsd,color,'micronbar','off','figSize','large')

%from here we need to compute:
%1) should have check for
%1)twin boundary length for all boundary types: parent, neighbor, twin-twin
%2)group merged grains by misorientation
%3)grain area
%4)schmid factor at parent/twin boundary for twin type

%with this data in arrays we can:
%1) apply voting scheme to segment structure and assign twin types
%2) compute twin thickness
```

```
%3) varient type



%reconstruction strategy is
isTwin=zeros(length(grains),1);
nmerged=length(mergedGrains)
for i=1:nmerged
    subgrains = grains(parentId == mergedGrains(i).id);
    maxArea=max(subgrains.grainSize);
    % need to group similar same orientations into single parent grain so
    % we compute the misorientation between mean components
    flag=false; cnt=1;
    while cnt <=length(subgrains) && flag==false
        mis=angle(subgrains(cnt).meanOrientation, subgrains.meanOrientation)./degree;
        area=sum(subgrains.grainSize(mis<5));
        if area >=maxArea
            isTwin(subgrains(mis>5).id)=1;
            flag=true;
        end
        cnt=cnt+1;
    end
end
% figure
plot(grains,isTwin,'micronbar','off','figSize','large')
hold on
% plot(grains.boundary)
plot(twinBoundary_tt1,'linecolor','w','linewidth',2,'displayName','Tensile Twin 1↙
Boundary')
hold off
```